Introduction

Machine Learning 2020
mlvu.github.io

machine learning

Admin

part 1:

What is Machine Learning?

Supervised ML: Classification

part 2:

Supervised ML: Regression

Unsupervised ML

What isn't Machine Learning?

your grade

exam	project
minimum grade 5.5	minimum grade 4.5
subject: Lectures, Literature, Homework	groups of 5
40 multiple choice questions	subject free see Canvas for suggestions
7 homework sessions (weekly)	project sessions (weekly, start next week)

attendance is only obligatory at the **project sessions** (at least one group member)

The final grade is the average of your exam grade and your project grade (weighed equally). The average should be above 5.5, and the exam should be above 5.5 and the project should be above 4.5.

reading

Reader: see links in syllabus

Meant to *support* the lectures. Almost all exam questions will be about the slides.

Recommended reading:
See canvas page. Not required, but may be helpful.

The literature for the course is a reader.

You may find the occasional reference to Peter Flach's book "Machine Learning". This was last year's book. We don't use it any more.

exam

Covers slides, supported by literature

40 multiple choice questions in three categories:
- recall
- applied knowledge
- active knowledge (calculating stuff)

Cheat sheet available for formulas

Practice exams (will soon be) available
check **mlvu.github.io** for last year's exams

first homework this week

thing by heart, but it shouldn't take you too long to look the answers up.

1 Linear Algebra

In all homework and lectures, bold lowercase letters like $\mathbf{x}$ indicate a vector, bold uppercase letters like $\mathbf{W}$ indicate a matrix and non-bold lowercase letters like x indicate a scalar (that is, a number).

Exercise 1

Explain in words what the following notations represent:

1. $f : \mathbb{R}^3 \rightarrow \mathbb{R}^2$
2. $\mathbf{y} = \mathbf{W}\mathbf{x}$
3. $\mathbf{z} = \mathbf{y}^T\mathbf{x}$
4. $\mathbf{W} \in \mathbb{R}^{5\times4}$

Hint: if you're not sure, see if you can find the symbols in this page: https://en.wikipedia.org/wiki/List_of_mathematical_symbols. Even

This is the first homework. If you find these symbols intimidating, or difficult to read, **make sure to show up to the homework session**. The mathematics of machine learning is relatively lightweight, but you do need to get used to the notation. If you have trouble with this sort of thing, the best thing to do is to face it early on and to ask us for help.

$$\begin{aligned}
\frac{\partial \text{loss}(w,b)}{\partial w} &= \frac{\partial \frac{1}{n}\sum_i (wx_i + b - y_i)^2}{\partial w} \\
&= \frac{1}{n}\sum_i \frac{\partial (wx_i + b - y_i)^2}{\partial w} \\
&= \frac{1}{n}\sum_i \frac{\partial (wx_i + b - y_i)^2}{\partial (wx_i + b - y_i)} \frac{\partial (wx_i + b - y_i)}{\partial w} \\
&= \frac{2}{n}\sum_i (wx_i + b - y_i)x_i \\
\frac{\partial \text{loss}(w,b)}{\partial b} &= \frac{\partial \frac{1}{n}\sum_i (wx_i + b - y_i)^2}{\partial b} \\
&= \frac{2}{n}\sum_i (wx_i + b - y_i)
\end{aligned}$$

This is one of the slides in Thursday's lecture. For some of you this may look trivial, and for some it may look scary. You don't necessarily have to understand all such slides to pass the exam, but you do, if you want to really understand machine learning.

The good news is that the math isn't as scary as it looks. You just need to get comfortable with a few basic ingredients. The first homework should help you do this. If you want to make the rest of the lectures easier to follow, **I strongly recommend spending some time with the first homework.**

project

Until February 27

- Explore. Practice.
- Choice of software is free. Worksheets for Python stack.

On February 27: pick a topic

After February 27

- Do experiments, write report.

You can of course, commit to the topic earlier, so you have more time to perfect the report, but we do recommend exploring a bit first.

project sessions

Every week, starting next week

Informal presentations, but one or two slides are required.

Report on your progress, *or lack thereof*.
Describe what problems you've run into. Discuss solutions with the group. and the TA.

Discuss what subjects you're considering.

topics we're considering

option 1: kaggle project: Predict disaster tweets
questions:
Is this doable with basic ML? Deep learning seems too ambitious.
What if we don't get good results?

option 2: code a neural network from scratch
Too ambitious? What would our experiments consist of?

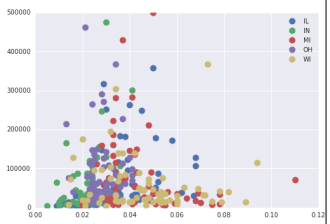
option 3: predict the results of football matches
where would we get the data?

Here's one example of what such a slide might look like.

first results

success:

Managed to load the data
First plot



problems:

The data is very unbalanced.
The data seems to overlap a lot. We're not sure if the features are informative.

If you've managed to load a dataset, a simple scatter plot of two of its features can be enough to get you a slide to present.

we have a bug :(

data loading bug:

```
Traceback (most recent call last):
  File "daily.py", line 4, in <module>
    Q = pd.read_csv(StringIO("""/cygdrive/c/temp/MQM Q.csv"""), header=0, usecols=["Date-Time"
  File "/usr/lib/python2.7/site-packages/pandas-0.14.0-py2.7-cygwin-1.7.30-x86_64.egg/pandas/
    return _read(filepath_or_buffer, kwds)
  File "/usr/lib/python2.7/site-packages/pandas-0.14.0-py2.7-cygwin-1.7.30-x86_64.egg/pandas/
    parser = TextFileReader(filepath_or_buffer, **kwds)
  File "/usr/lib/python2.7/site-packages/pandas-0.14.0-py2.7-cygwin-1.7.30-x86_64.egg/pandas/
    self._make_engine(self.engine)
  File "/usr/lib/python2.7/site-packages/pandas-0.14.0-py2.7-cygwin-1.7.30-x86_64.egg/pandas/
    self._engine = CParserWrapper(self.f, **self.options)
  File "/usr/lib/python2.7/site-packages/pandas-0.14.0-py2.7-cygwin-1.7.30-x86_64.egg/pandas/
    col_indices.append(self.names.index(u))
ValueError: 'Value' is not in list
```

what we've tried:
- The error doesn't show up in google.
- CSV file does load in Excel, google sheets

If you get stuck somewhere, just present the problem, and what you've tried so far. Maybe some students have encountered the same problem, maybe the TA can give you some tips for how to solve the problem.

You don't always have to present progress, but you do have to do something every week, and present what you've done.

our code so far

```
def sigmoid(Z):
    return 1/(1+np.exp(-Z))

def relu(Z):
    return np.maximum(0,Z)

def sigmoid_backward(dA, Z):
    sig = sigmoid(Z)
    return dA * sig * (1 - sig)

def relu_backward(dA, Z):
    dZ = np.array(dA, copy = True)
    dZ[Z <= 0] = 0;
    return dZ;
```

13

If you're halfway through some complex coding, you can present some code that's finished, and explain what it does.

warning

If you don't present, **with slides**, it counts as being absent, which will hurt your grade, or cause you to fail the project (in extreme cases).

14

We won't fail you for being absent (or not presenting) just once, but we may lower your grade. If you miss *all* group meetings or never present, we may choose to fail you for the project.

a word on group dynamics

Some people don't like working in groups, so they turn a project assignment into five individual assignments by immediately breaking up the work and never meeting again. Not only is this not the point of group work, it's very likely to fail: if one group member doesn't deliver, or misunderstood the idea, the whole project goes wrong.

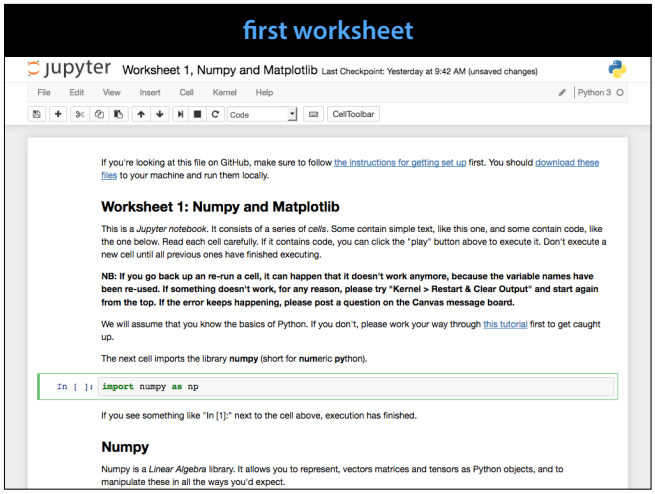
a word on group dynamics

- **don't**: divide duties, split up, and never meet again
- **do**: meet regularly, discuss level of ambition
- **don't**: pick a topic right away
- **do**: spend time to make sure you're on the same page
- **don't**: worry too much about equal workloads
- **do**: each do all the worksheets individually

16

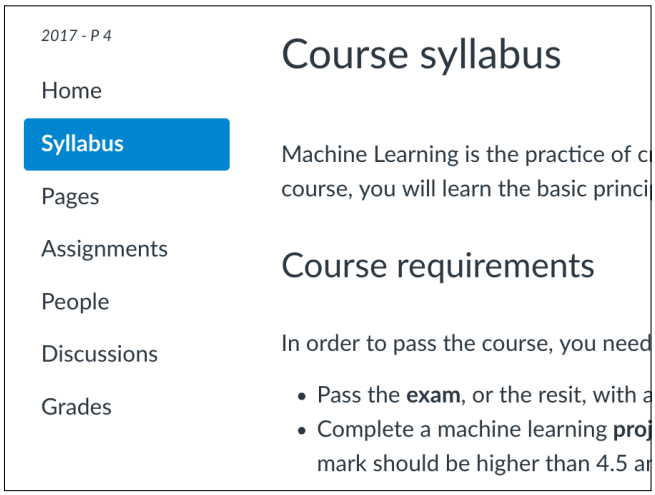
We have a very diverse group of students, and many will be grouped with people you haven't met before. Make sure that you're on the same page about the kind of project you'd like to do and how complicated you want to make stuff. Nothing kills a project faster than a single highly motivated student dragging everybody into a hugely ambitious project.

To make sure that everybody is clear about the group's goals, take your time before you pick the topic. Meet every week and do lots of exploratory work. give everybody time to learn what machine learning is about, and to join the discussion. It helps if everybody does the worksheets on their own, so you have a common experience to build on.

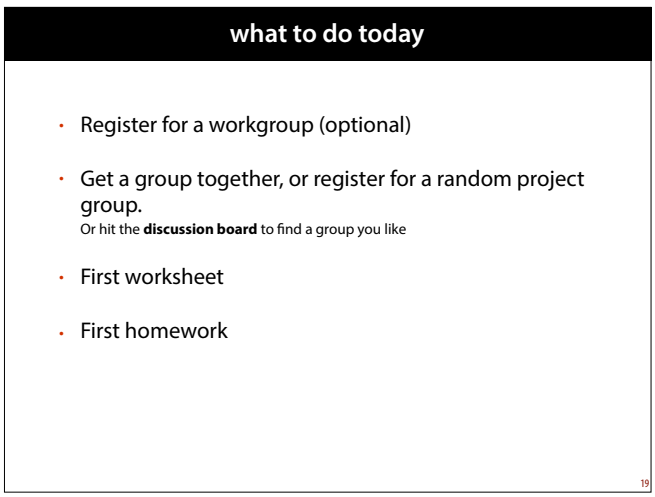


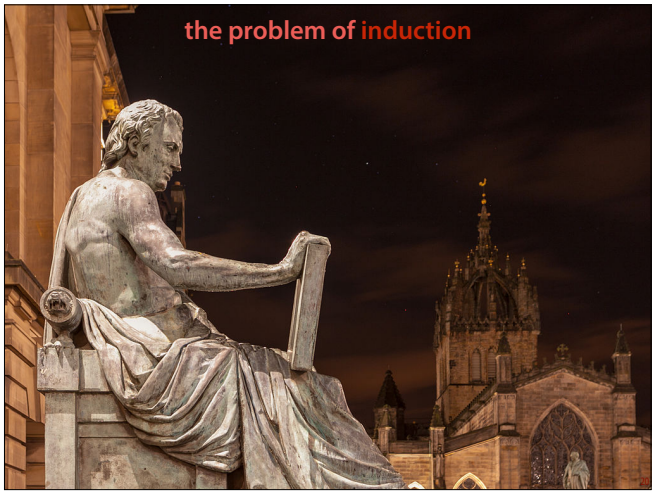
The worksheets are purely there to help you get started. You don't have to do them if you don't want to, and you can use any software you like.

However, the software presented in the worksheets is very much the standard machine learning/data science stack, so if you expect touse what you learn here more, we recommend using the software presented in the worksheets.



You can ask me any question you like but only after yo've read this page from top to bottom. There is an FAQ section.





The problem of how we learn (and how we can make machines that learn) is an instance of the *problem of induction*. It was first posed as a problem by 18th century philosopher David Hume (pictured).

Inductive reasoning is essentially a philosophical name for learning. We observe something happening a number of times, so we *infer* that it'll probably happen again the next time. We're not absolutely certain, and it doesn't follow logically, but we're sure enough to use that knowledge to our advantage.

This is very different from the *deductive reasoning* which philosophers had studied since antiquity. Deduction is rule-following. It's what computers do best. In order to make computers do something like inductive reasoning, and in order to fully understand how *we* do it, we need to reduce it to rules. But Hume argued that inductive reason can not be proved to

work by deductive methods.

Image source: By Bandan - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=29332424

---

deductive reasoning	**inductive reasoning**
All men are mortal Socrates is a man	The sun has risen in the east every day of my life
therefore Socrates is mortal	so it will do so again tomorrow
discrete unambiguous provable known rules	fuzzy ambiguous experimental unknown rules

21

For deductive reasoning, we know the rules, and we understand them perfectly. For inductive reasoning the rules are not so clear. For instance, whenever I visit a funeral, I'm never the person being buried. Therefore, the more funerals I visit, the more certain I should be that next time it won't be my funeral. Clearly this is not the case (usually the opposite is true).

So, if inductive reasoning doesn't follow as a special case of deductive reasoning, and inductive reasoning applies sometimes and it doesn't at other times… how do we do it? Why is the funeral example obviously wrong, and the sun example obviously right? If inductive reasoning cannot be reduced to deductive reasoning, do we have any hope of reducing it to a computer program?

In many ways, the problem of induction is still unsolved. We can teach computers to learn pretty well these days, but we still don't fully understand what all the rules are.

---

a broad definition

(from expertsystem.com)

Machine learning […] provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

22

Here is a decent definition of Machine Learning that covers most of he important aspects. It's a system (i.e. a computer), it improves its behaviour based on experience, and the resulting behaviour has not been explicitly programmed.

Before we look at what this means in practice, let's have a look at some basic questions about machine learning:

- Where do you use it?
- When do you use it?
- What does it solve?

quote source: http://www.expertsystem.com/machine-learning-definition/

where do we use ML?

Inside other software

Unlock your phone with your face, search with a voice command,

In analytics, data mining, data science

Find typical clusters of users, predict spikes in web traffic

In science/statistics

If any model can predict A from B, there must be some relation

23

---

what makes a suitable ML problem?

- We can't solve it explicitly.
- Approximate solutions are fine.
- Limited reliability, predictability, interpretability is fine.
- Plenty of examples available to learn from

bad	good
Computing taxes	Recommending a movie
Clinical decisions	Clinical decision *support*
Parole decision (support)	Predicting driving time
Unlocking phone	Recognising a user

24

Fun fact, ML systems do exist for parole decisions, and face detection is used for unlocking many phones. These are problematic developments.

Why don't we have a good explicit solution?

- Explicit solutions are expensive.
- The best solution may depend on the circumstances.
- The best solution may change over time.
- We don't know exactly how our actions influence the world.
- We can't fully observe the world.

25

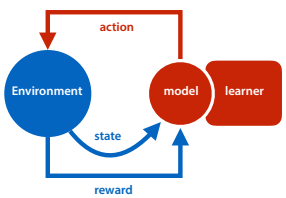
Machine Learning may solve any or all of these problems. Depending on which problems you want to solve,

---

the broad view: an intelligent agent

online learning: acting and learning at the same time.

reinforcement learning: online learning in a world based on delayed feedback.



coming in week 7

26

If you have all these problems, i.e. you want to train a self-learning agent that continuously adapts and can freely move throughout a world, based on an occasional reward signal, you will need a very broad framework to model the problem. In **online learning**, we are choosing actions based on inputs, and also learning form those inputs at the same time.

In **reinforcement learning**, things are made even more complicated: we interact with a world that also gives us a reward signal. We try to maximise the reward, but we don't know which action lead to which reward.

Dealing with all these problems at once is very complicated. In most cases, we don't actually need to learn online. And instead of a reward signal, we have explicit examples of which input maps to which output. In such cases, we can simplify the problem a lot.

offline learning

Separate *learning* and *acting*

- Take a fixed dataset of examples (aka *instances*)
- Train a model to learn from these examples
- Test the model to see if it works
  If the model works, put it into production

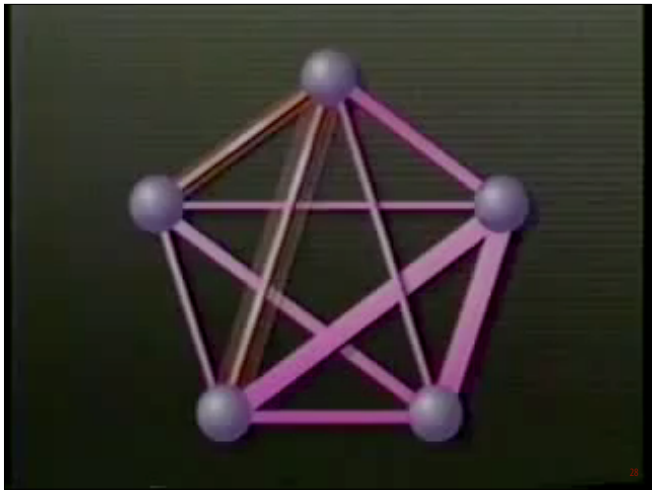
most of the course

27

In **offline learning** you separate the acts of learning of a model and of putting a learned model to use. You gather a dataset of examples beforehand, you train a *model,* test it, and once you're sure it works well enough, you use that version of the model (for instance by sticking into a computer program). The finished program will never learn while it's running. There is no feedback loop between learning and running.

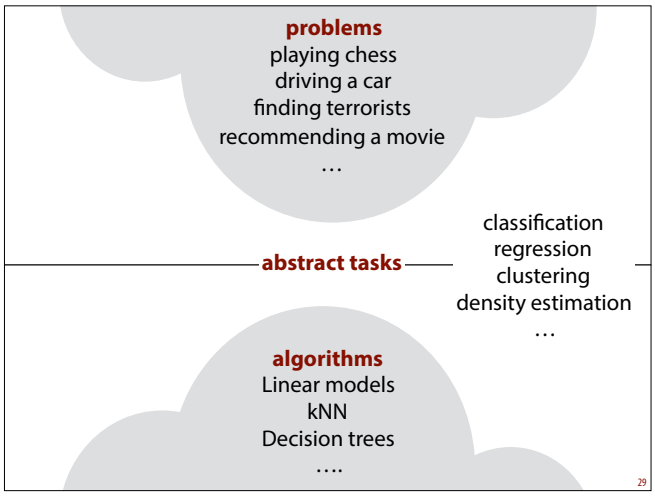
While this robs the exercise of some of it's more exciting aspects (it's a far cry from building androids), it still allows us to do something very useful: it allows us to learn programs that we have no idea how to write ourselves. For instance, we can learn a program that detects birds in pictures: we have no idea what kind of rules would be required, or how to design an algorithm to do it. Machine learning allows us to create such a program from a set of examples.

Almost all of this course will focus on offline learning.



This video illustrates the basic idea: feed the computer the examples one by one, and tell it the target value for each.

video source:
https://www.youtube.com/watch?v=7BtLqqJVP9w
https://archive.org/details/perceptron_documentary_excerpt



Now, the main problem with machine learning is that **we want solutions that are applicable across domains**. You don't want to dedicate your entire life crafting a perfect self-learning computer chess program, and then find out that your ideas have no use for anything else. We want to solve the problem of machine learning **in general**: instead of studying each problem in isolation, we want solutions that can be applied to many problems.

To make this possible, machine learning is usually built on one of a few **abstract tasks** like *classification*, *regression* or *clustering*. If you have a practical problem, like chess playing, you find a way to abstract the problem of playing chess (or part of it) to the generic task of, say, classification, and then you pick one of many existing classification *methods*.

abstract tasks

supervised	unsupervised
Explicit examples of input *and output*.	Only *inputs* provided.
Learn to predict the output for an unseen input.	Find any pattern that explains something bout the data.

30

Abstract tasks come in two basic flavours: **supervised**, and **unsupervised.**

In supervised tasks, we have explicit examples of both inputs and the corresponding outputs. What we have to learn is the program that maps any input to the corresponding output. For instance, we may be provided with emails and given a label **spam** (advertising) or **ham** (genuine) for each. The task then, is to train a program to assign these labels to new emails.

In unsupervised tasks, there is no target value, only the data. All we can do then is to learn some structure in the data. For instance we can cluster students to see if there are natural groups, or see if we can detect which financial transactions are "unusual".

---

supervised learning tasks

Classification: assign a class to each example.

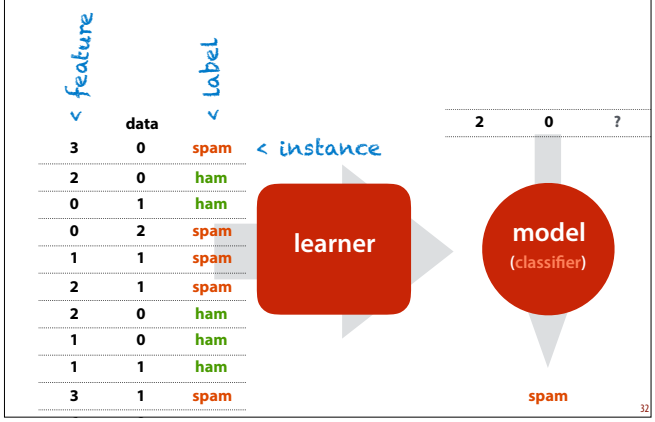
Regression: assign a number to each example.

31

These are the two main forms of **supervised offline learning**.

---

classification



32

This is the basic framework of classification. The data that we provide our system with consists of examples, called **instances**, of the things we are trying to learn something about. In this example, our instances are e-mails.

We must then make a series of measurements about each instance. In the case of e-mails, we may measure how often a specific word occurs. The things we measure are called the **features** of the instance. We can measure numeric features (like age or speed), but they can also be categoric (like gender or color).

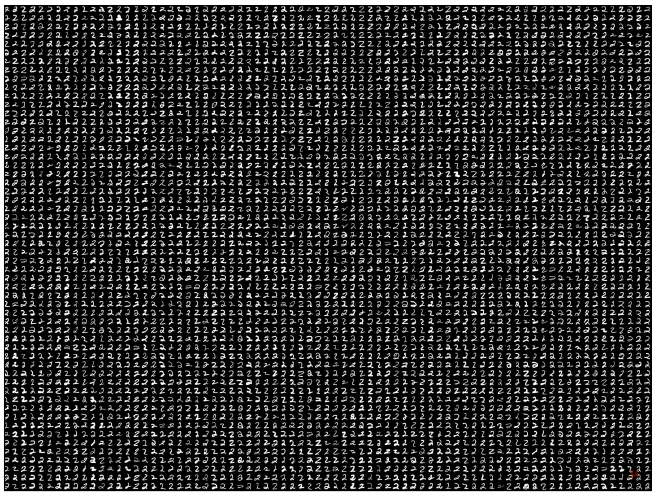
Finally we have the **target value**: the thing we are trying to learn. In classification, this is always a categoric value, or a *class*: one of a handful of possible values. In this case, is the e-mail spam (an unwanted advertising e-mail), or ham (a genuine e-mail).

This dataset is then fed to a learning algorithm. This can be anything, but it has to produce a classifier. A classifier is a small "machine" that (attempts) to solve the learning problem. That is, it takes a new instance, one that wasn't in the original dataset, and for which we don't know the target value, and it makes a guess at the target value.

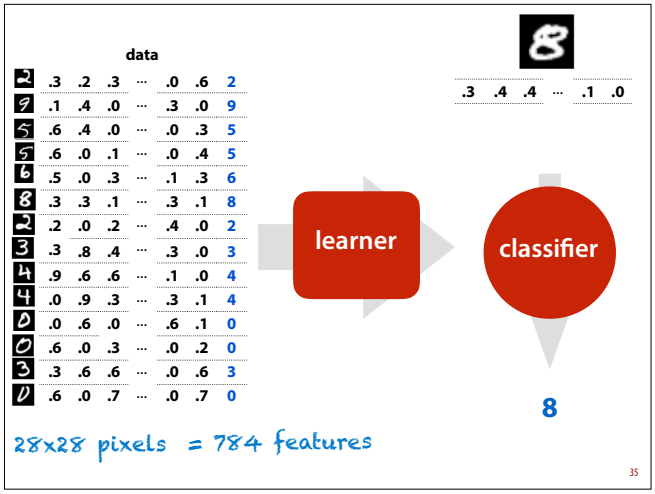
Let's look at some examples of how we can reduce real-world problems to classification. We'll start with **handwriting recognition**. Specifically, reading a ZIP code on an envelope. This involves many difficult problems: aligning the envelope, finding the address, finding the ZIP code within the address, segmenting the address into digits, etc.

Our first aim is to reduce the problem to a simple classification problem: we will assume that we are given an image of a *single digit*, and the task is to predict what the digit is. This is a much simpler problem, but still a challenging one.The next step is to gather some training data we can learn from.

image source: **https://rafalab.github.io/dsbook/introduction-to-machine-learning.html**



This is a small selection from the MNIST dataset: 60 000 examples of handwritten digits. This translates very simply to classification: each picture of a digit is an **instance**, and the **target** is one of ten classes: 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.

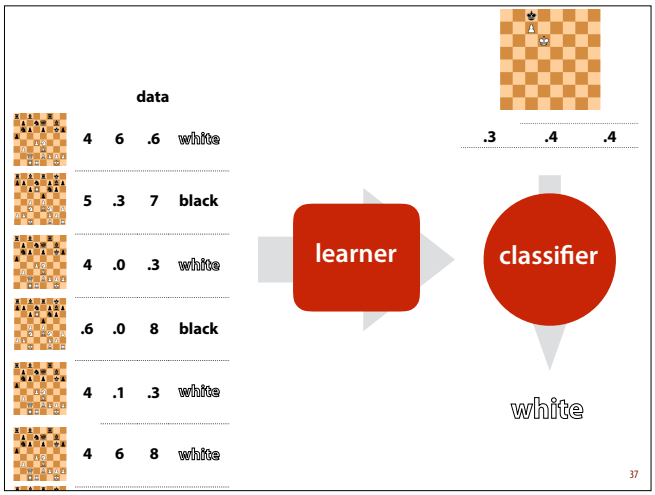


A simple way of attacking this problem is to make each pixel a feature. Here's what that looks like. For each instance, we translate each pixel to a value between 0 (black) and 1 (white). This gives us instance with 784 features each, labeled with a digit from 0 to 9. We feed these to the learning algorithm, which produces a classifier. We then get a new example, and ask the classifier what it thinks. Once we have a classifier that does well, we can use it in a larger system, for recognising digits.

The current best performing classifier for this task has a **probability of 0.21%** of getting an unseen example wrong.

Note that we haven't fully solved the problem of character recognition. We still need to cut a sequence of digits into individual digits, feed those digits to the classifier and process the results. This all the work we have to to to translate our **real problem** to the **abstract problem** of classification. Machine learning has solved part of the problem for us, but there is usually still a lot of engineering left to do.

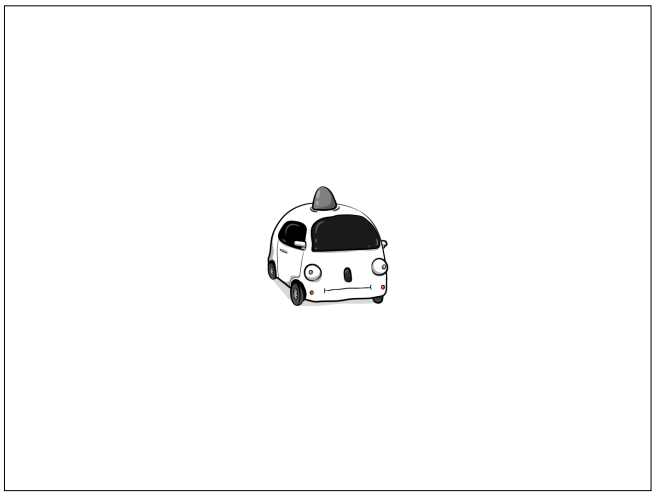
Let's look a problem that's a bit further removed from classification: game playing. Specifically, playing chess. How do we abstract the problem of playing chess to a classification problem? The trick here is to make things easy for ourselves, by only abstracting *part of the problem*. We won't solve the whole thing with machine learning, but we'll learn a function that'll be useful.



For instance, we take a database of chess games, and label each position with the winner of the corresponding game. The aim is to predict, for given a position, which player is going to win the game.

We could turn such a classifier into a chess player, by searching for positions from which we are likely to win. Perhaps you are familiar with the minimax algorithm: you could easily such a classifier as a *heuristic* in minimax.

A difficult problem here is *which features to use*. One option is to report how much of each black and white piece is left, which would allow at least some positions to be predicted accurately, if one player has a strong material advantage. For more insightful learning, we need better features. **Domain expertise** can often be translated to good features: are there passed pawns, rooks on an open file, does a player own both bishops, etc.



Last example, a self driving car. How do we turn the problem of making a self driving car into a classification?

image source: the oatmeal, ***http://theoatmeal.com/blog/google_self_driving_car***

ALVINN (1995)

.3	...	.3	right
.0	...	.0	left
.0	...	.0	left
.1	...	.2	right
.3	...	.4	straight
.1	...	.6	left
.2	...	.3	right
.4	...	.4	left
.6	...	.6	straight
.3	...	.3	straight
.0	...	.0	left
.3	...	.3	left
.6	...	.6	straight
.3	...	.1	right

960 features

39

Here is an actual self-driving car system from 1995. They used a very low-resolution, black-and-white camera to film the road, and observed a human driver's action to label each frame with an action. As with the digit reconition example, we simply make each pixel a feature.

This system actually drove from coast-to-coast autonomously in America (albeit with a human driver executing the system's instructions). NB: The actual system had more than three actions to allow for more gentle steering.

---

problems
playing chess
driving a car
ocr

classification

algorithms
?

40

This is what I meant by translating problems into abstract tasks. By translating all these problems into classification problems, we can now apply any **classification algorithm** and see how well it does.

So how do we fill in the other half of this picture? Once we have a classification task, with features selected and a set of good examples, how do we actually produce a classifier?

We'll look at three simple examples: a **Linear classifier,** a **Decision Tree classifier** and a **Nearest Neighbors classifier**. We'll only explain them briefly to give you a sense of how these problems might be solved. Don't worry if you don't totally get it yet. All methods will be discussed in more detail in later lectures.


181	46	male
181	50	male
166	44	female
171	38	female
152	36	female
156	40	female
167	40	female
170	45	male
178	50	male
191	50	male

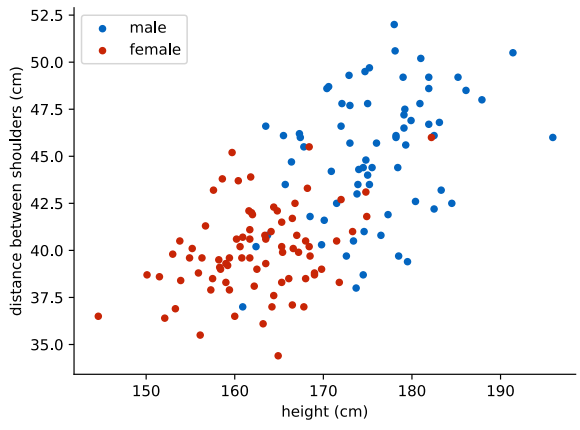
41

Here's a dataset that we'll use as a running example. Its **instances** are US soldiers, the two **features** are the height (or 'stature') and the distance between the shoulder blades ('interscye'). The class is their biological sex. Are these two values enough to predict whether somebody is **male** or **female**?

data source: ANSUR II
image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011

---

feature space

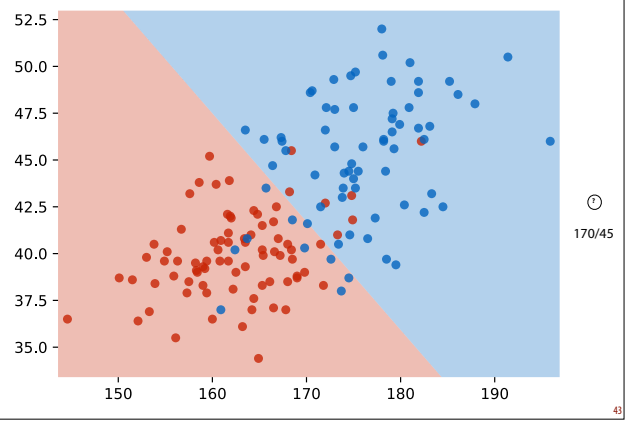


42

Since we have only two features, we can easily plot our dataset.

We call this space, where every instance is a point, the **feature space**. If we had 3 features, it would be a 3D space. For higher numbers of features, we may have difficulty visualizing the feature space, but that shouldn't stop the classifier: any classification method we come up with should, in principle, work on an arbitrary number of features.

example 1: linear classifier



43

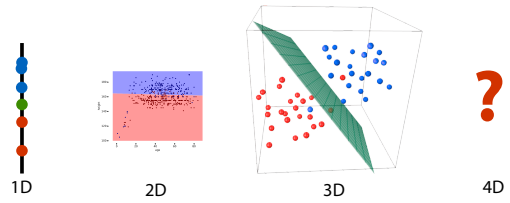
Here is a simple idea for a classifier: *draw a line*. We just draw a line, and call everything above the line **male**, and below it **female**.

As you can see, a few examples end up misclassified, but most of them are on the correct side of the line. Our classifier would already do much better than one that would simply guess at random or call everything **female**.

If we see a new person, all we need to do is measure them, and see whether they end up above or below the line.

---

hyperplanes



44

An important thing to note is that "drawing a line" is a technique that only works in two dimensions (i.e. if we have two features). Our methods need to work, at least in principle, for whatever number of features we decide to use. The more generic version of the idea to "draw a line" is to cut the feature space in two using a line-like shape.

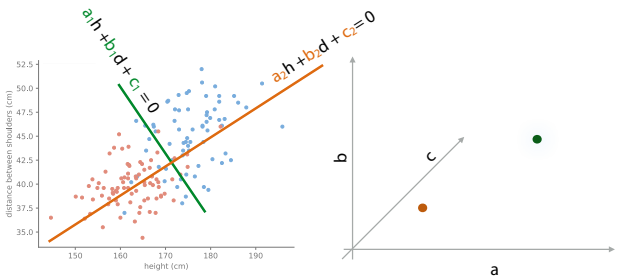
In 1D, the equivalent structure is a point. Anything above the point is male, anything below it, female.

In 3D, we can cut the feature space in two with a plane.

In four or more dimensions, we can no longer draw it intuitively, but luckily the mathematics are very simple. We'll see how to define this in the next lecture.

---

the two spaces of machine learning



feature space (2D) model space (3D)

45

Which line should we choose? We can visualise this problem in the feature space. In the feature space (or instance space), each instance is a point, and our current classifier is a line.

Since a line is defined by two parameters (the slope **s** and the intercept **b**), we can visualise the **model space** as a plane. To search the model space, we define a **loss function** which tells us how well a particular model does for our data.

---

loss function

$\text{loss}_{\text{data}}(\text{model})$ = performance of model on the data
(the lower the better)

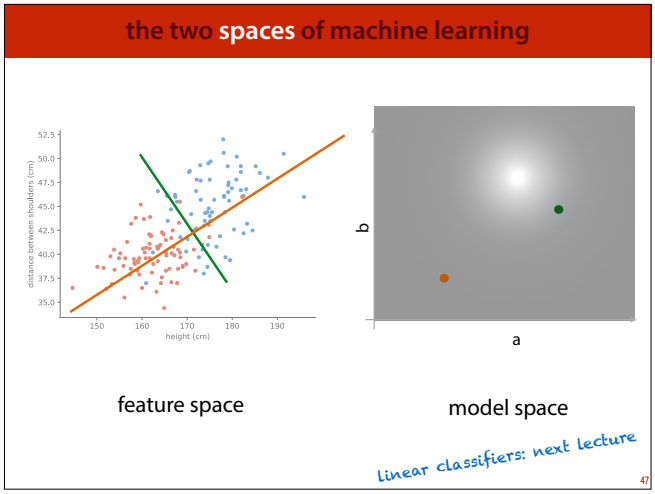
for classification: e.g. the number of misclassified examples

46

To capture which model we prefer, we define a loss function. The lower the loss, the better the model.

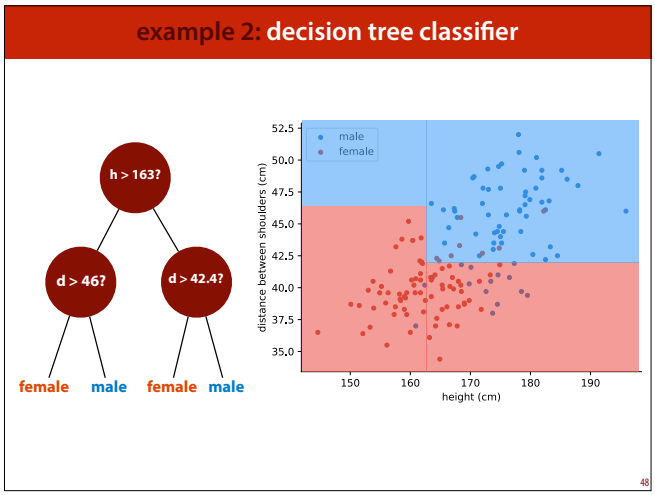
Note that the loss function has the **model** as input and the **data** as a constant (as opposed to the model itself, which is a function of the data).

The best loss function to use for classification is a complex question. We'll come back to that later.



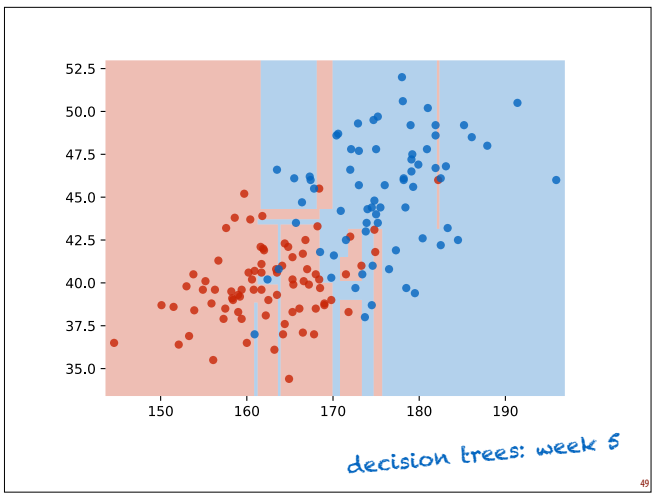
Once we have a loss function, we can colour our model space with the *loss of each model* (for our current data). The brighter, the better.

All we need to do now is find the brightest point, which corresponds to the best model. More on that next lecture. In this case we can see which the brightest point is, but remember that the model space is usually high-dimensional.



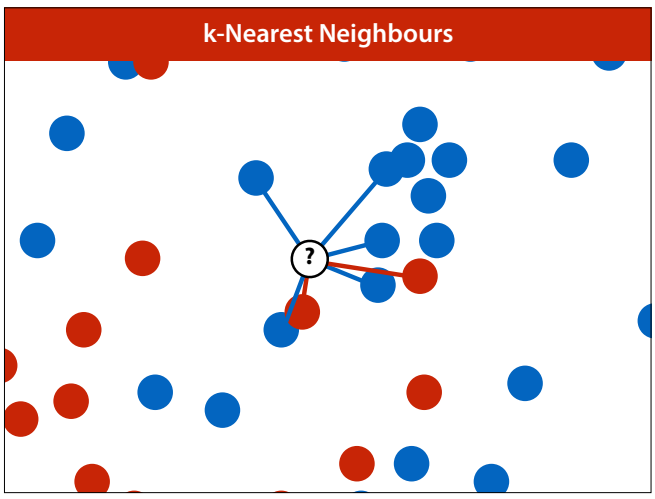
Here is a radically different approach: **a decision tree**. This classifier consists of a tree, which studies one feature in isolation at every node. In this case, it moves left if the feature is lower than some threshold value, and right if the feature is higher.

The model space of all possible decision trees is a lot more difficult to visualize. Often, decision trees are "grown" by adding nodes from the root until a particular criterion is reached. We'll discuss how to train decision trees in detail in week 5 (the algorithm is simple, but it requires a little probability theory).



Here's what the actual decision tree algorithm from sklearn does (with default settings).

In week 5, we'll see how this learning algorithm actually works.

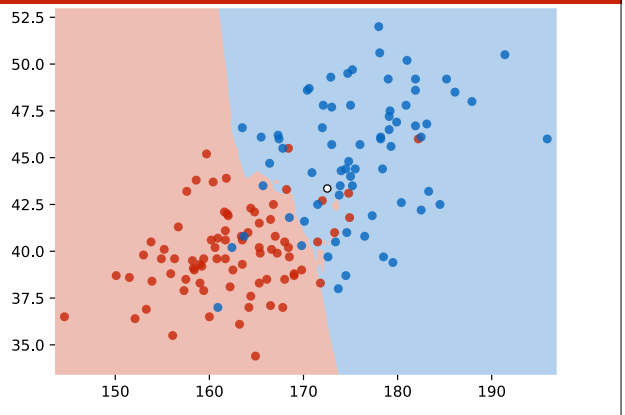


Finally, here is an example of a l*azy* classifier. k-Nearest neighbours. It doesn't do any learning. It just *remembers* the data.

For a new point (indicated by the question mark), it just looks at the *k* points that are closest (k=7 in this picture), and assigns the class that is most frequent in that set (blue in this case).

k is what we call a **hyperparameter**: you have to choose it yourself before you use the algorithm. We'll discuss how to choose hyperparameters next week.

k-Nearest Neighbours



51

Here's what the decision boundary looks like for k=7. The point from the previous slide is indicated in white.

variations

- Features: usually numerical or categorical.
- Binary classification: two classes
- Multiclass classification: more than two classes
- Multi*label* classification: none, some or all classes may be true
- Class probabilities/scores: the classifier reports a probability for each class.

52

A few variations are possible on this basic scheme. Firstly, the features are usually numerical or categorical. Some models can handle only numerical, in which case, any categorical features have to be translated to numerical ones (we'll see how to do that next week).

Binary classification (a task with two classes) is probably the simplest and most well-studied task. If you have more than two classes, some classifiers (like kNN) can deal with that without a problem. For others (like linear classification), you'll need to find clever way to turn a binary classifier into a multiclass classifier.

Multilabel classification is a much more complex task. We won't go into it in this course, but it's an active subject of research.

Instead of a single verdict, it can often be helpful if a classifier assigns a **score** to each class. If we want a single class, we pick the one with the highest score, but we can also check what the second most likely class is, and how sure the classifier is of its verdict. This is often important if the consequences of of a wrong classification are very serious (i.e. deciding whether to operate on someone, or whether to investigate someone for criminal activity).

recap

- **Offline learning:** train once, use model.
- **Abstract tasks:** generalize the problem of learning
- **Supervised learning:** examples of input *and output*
- **Classification:** supervised learning with categorical output
- **Classification algorithms:** linear, decision trees, kNN
- **ML concepts:** feature space, model space, loss function

53

A few variations are possible on this basic scheme. Firstly, the features are usually numerical or categorical. Some models can handle only numerical, in which case, any categorical features have to be translated to numerical ones (we'll see how to do that next week).

Binary classification (a task with two classes) is probably the simplest and most well-studied task. If you have more than two classes, some classifiers (like kNN) can deal with that without a problem. For others (like linear classification), you'll need to find clever way to turn a binary classifier into a multiclass classifier.

Multilabel classification is a much more complex task. We won't go into it in this course, but it's an active subject of research.

Instead of a single verdict, it can often be helpful if a classifier assigns a **score** to each class. If we want a single class, we pick the one with the highest score, but we can also check what the second most likely class is, and how sure the classifier is of its verdict. This is often important if the consequences of of a wrong classification are very serious (i.e. deciding whether to operate on someone, or whether to investigate someone for criminal activity).

image source: ***https://twitter.com/archillinks/status/1022889384494940160***

---

regression

classification: output labels are classes (categorical data)

regression: output labels are *numbers*

55

---

regression

data		
3	0	1.1
2	0	4.5
0	1	12.3
0	2	5.1
1	1	9.1
2	1	1.2
2	0	5.2
1	0	6.1
1	1	1.9
3	1	1.8
6	0	3.2
0	1	0.1
0	0	0.4

learner

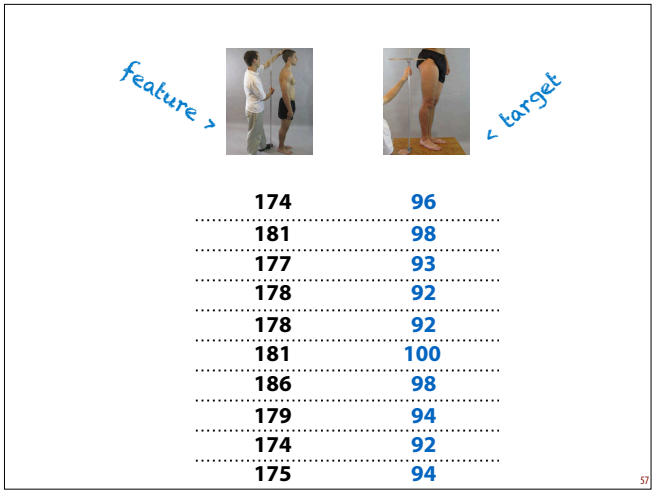
2	0	?

model

1.8

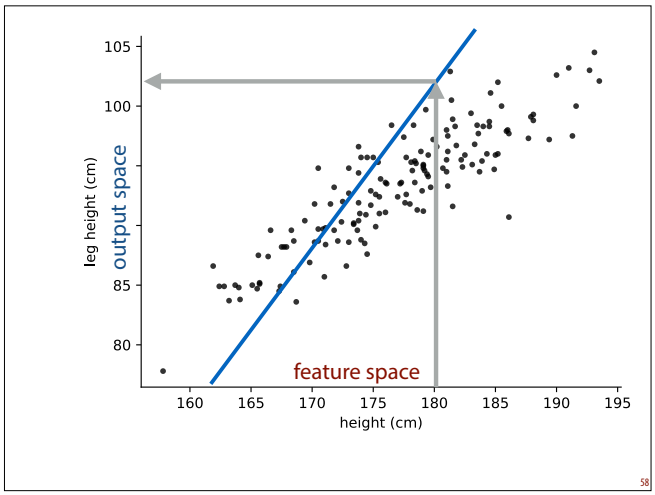
56

If your target value is expressed as a numeric value, we call it **regression**. It works exactly the same as classification, except we're predicting a number instead of a class. That is, the model we're trying to learn is a function from the feature space to $\mathbb{R}$.



Staying in the same domain, here is another dataset. Note that we have **only one feature** this time. The other numerical column is the **target** data. For general regression we should be able to handle datasets with arbitrary numbers of features.

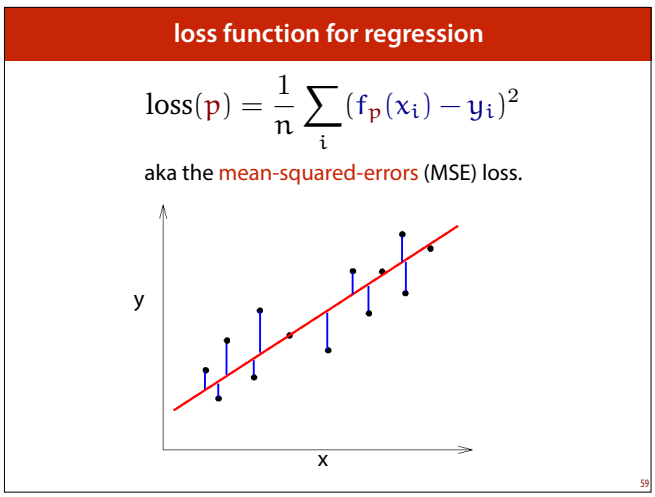
data source: ANSUR II
image source: MEASURER'S HANDBOOK: US ARMY AND MARINE CORPS ANTHROPOMETRIC SURVEYS, 2010-2011



Here's what our data looks like. Note that though it looks the same as in the classification example, this time we're plotting both the outputs and the feature space in the same figure.

We can use a **linear model** again. But note how differently we're using the model. Previously, we wanted to segment the feature space into two classes. Now we're trying to model the relation between the feature(s) and the target.

Now, the line I've drawn here isn't very good. It predicts much too high a value. To determine how good a model is, we must choose a **loss function**.



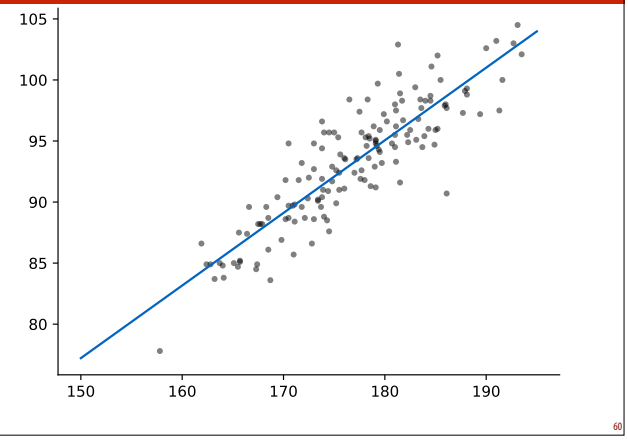
The **loss function** maps a model to a number that expresses how well it fits the data (the smaller the loss, the better). Offline machine learning boils down to finding a model with a low loss for your data.

Here is common loss function for regression. **p** stands for the parameters that define the line. We simply take the difference between the model prediction and the target value from the data. This is called a **residual**. We square, and then sum all residuals.

Sometimes we also divide by the size of the data (n). Occasionally, we multiply by 1/2 for technical reasons. This changes the actual value, but not *which* model produces the lowest loss, which is the question we want to answer.

*image source: **http://cs.wellesley.edu/~cs199/lectures/35-correlation-regression.html***

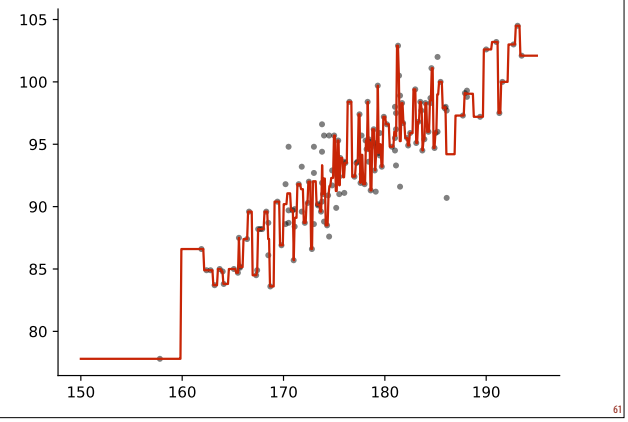
linear regression



60

This is the line with the lowest MSE loss for this data.

Regression tree

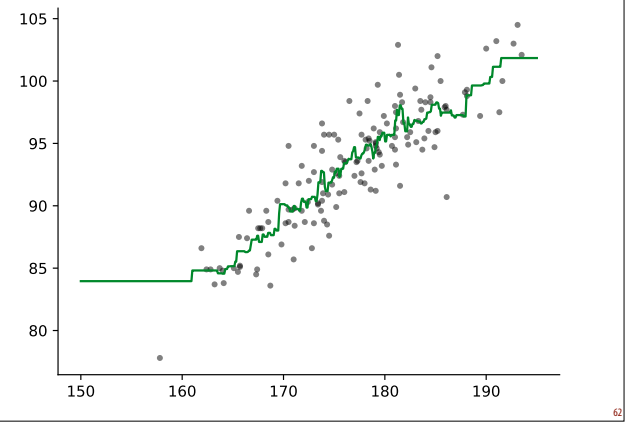


61

We can also use the **decision tree** principle to perform regression. We simply segment the feature space into blocks, using a tree as before, and instead of assigning each a *class*, we assign each a *number*. This model covers the data very well, for most points in our data it predicts exactly the right value. Does this make it a rally good model?

---

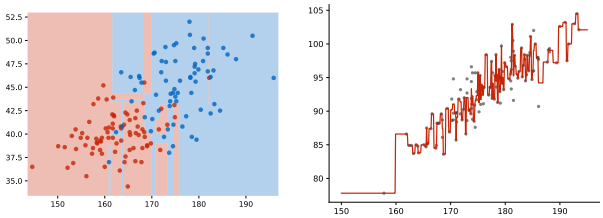
kNN regression



62

For the sake of completeness, here is what kNN regression looks like. Its prediction is the average of the k nearest points in the data (k=7 here).

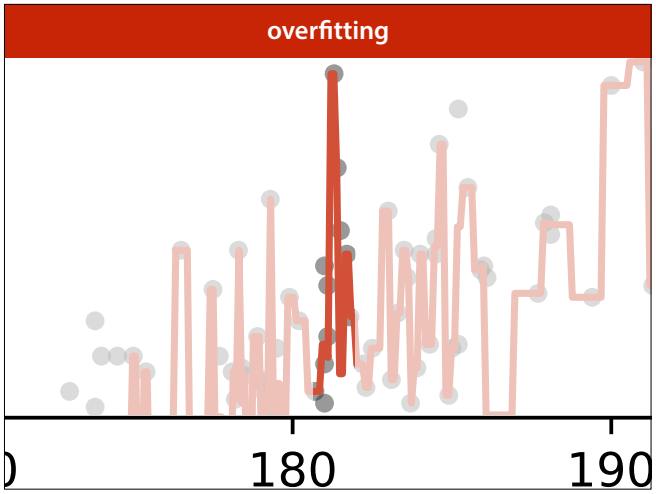
Overfitting



These models achieve much lower loss (Accuracy/MSE) than the linear or kNN variants. Would you use them?

63

Bear in mind that you usually *can't look at your model like this*. You'll often have hundreds of features, and all you'll have to go on is the **training loss**: the los for a given model on the training data.

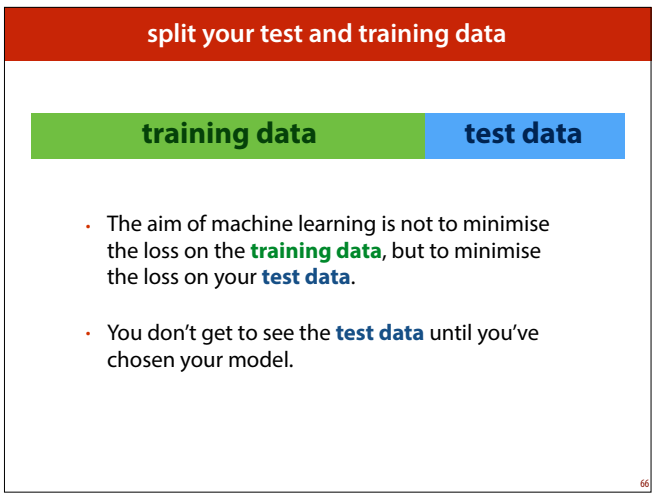


Let's look at this spike: the model seems to be convinced that people who are 183cm tall have proportionally much longer legs than people who are 182 or 183 cm tall. This isn't true of course, it just happens to be the case that in this area there was one person in the data with long legs (and no other people). The model is fitting details of the dataset that are *random noise*.

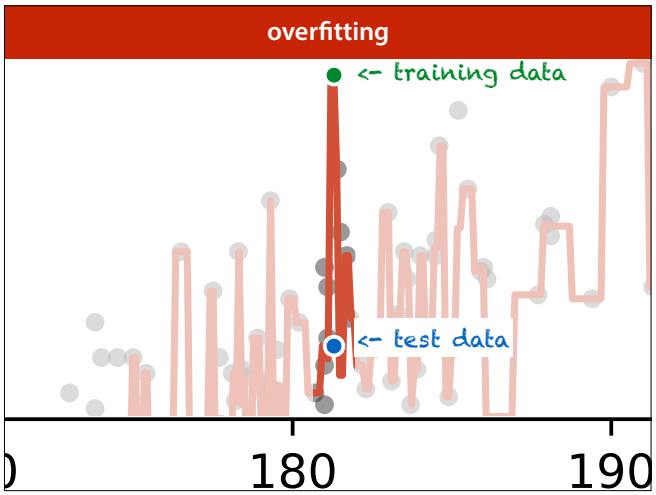
When a model overfits, we sometimes say that it is **memorizing** the data, when it should be **generalizing**.

This is the most important rule in machine learning. The regression tree had the lowest loss on the training data, **but it's actually the worst model**.

It means nothing how many of the training instances the model gets right. What we actually want is a model that does well on new data; data that it hasn't been trained on.



The simplest way to check this is to **withhold data**. You keep some data hidden from the model, and then check how well a particular model does on this part of the data. The data you show your model is called **training data**, the data you withhold is called **test data**.



Now we see that the regression tree is a terrible model. The training data may show a spike at 183, but the test data will just follow the linear pattern.

generalization

The problem of machine learning is to choose a model that fits the *pattern*, and ignores the *noise*.

68

There is no general rule for what constitutes noise and what constitutes a pattern. for one dataset the linear regression may be best, and for another, the regression tree.

This brings us back to the problem of induction, sometimes we can learn from past experience (like whether the sun will rise again tomorrow) and sometimes we can't (like who will be in the coffin at the next funeral). When you can learn, and which models work, is an entirely open problem.

in code (cf worksheet 2)

```
# Choose a model: Decision Trees
from sklearn.tree import DecisionTreeClassifier

x_train, x_test = … # matrix of features
y_train, y_test = … # target class labels

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train) # Search for model

# classify new data
y_predicted = tree.predict(x_test)
# see how well we do
print accuracy_score(y_predicted, y_test)
```

69

Here's what the basic recipe looks like in code (using the `sklearn` library). Note that the actual machine learning happens in just two lines of code.

All you need to do is decide your features, and decide your target values (classes in this case). Once you've done that, you're doing machine learning in two lines of code. You can then test how well your model does (more about that in week two) and keep trying different models until you get the performance you're happy with.

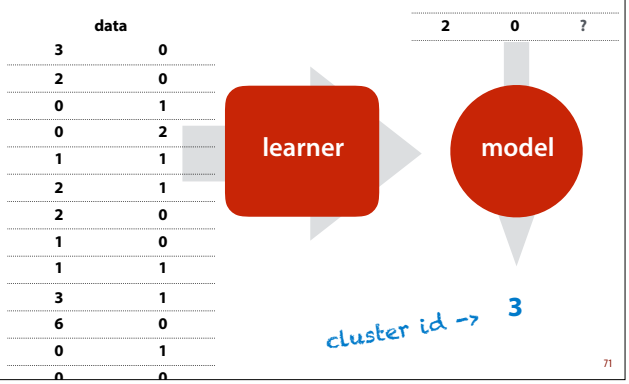
unsupervised learning tasks

- Clustering
- Density Estimation
- Generative Modeling

70

In regression and classification we had a set of input/output examples: we had **labeled data**. In other words, we could show our learner exactly what we want it to learn. In **unsupervised learning**, we don't have that luxury. We just have the data, and we want the learner to expose some kind of structure in it.

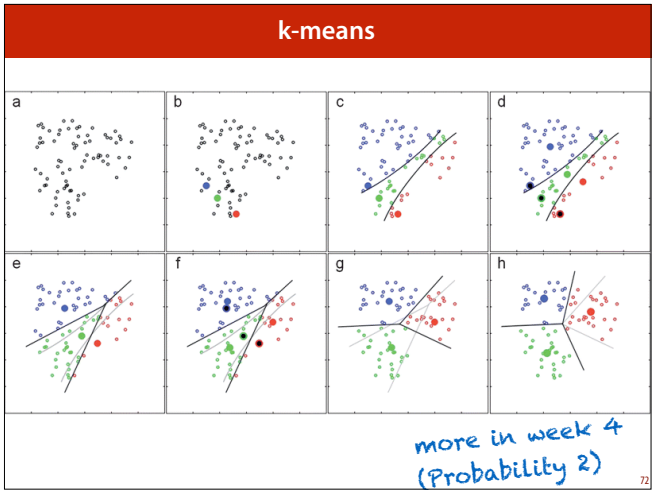
clustering



71

In the case of clustering, we ask the learner to split the instances into a number of clusters. The number of clusters is usually given beforehand by the user.

This looks a lot like classification, but note that there are no example classes provided by the data.

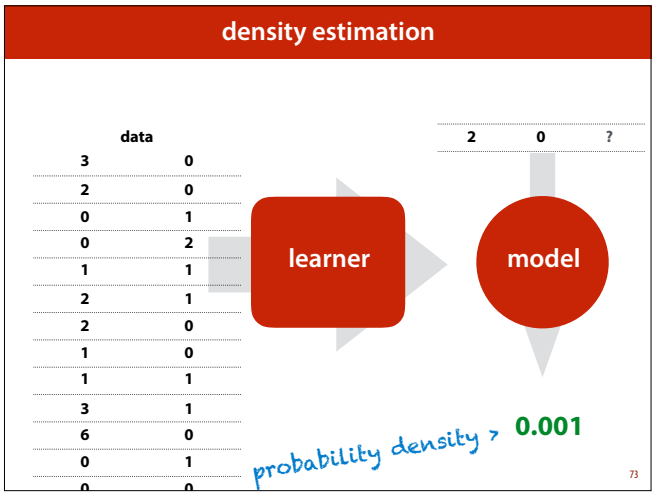


Here is one example of a simple clustering algorithm.

This algorithm is called **k-means** (not to be confused with kNN). In the example we will separate the dataset shown in (a) into three clusters.

We start (b), by choosing three random points in the feature space (the red, green and blue points), called the "means". We then colour every point in the colour of the nearest mean.

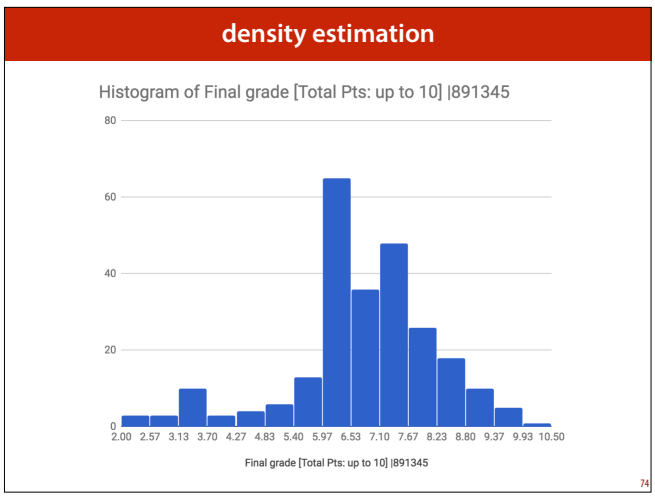
The second step is to remove the original means, and recompute them as the actual means of the sets of red, green and blue points. We then iterate these two steps, alternately recomputing the means and recovering the points, until the algorithm converges to a stable state.



In density estimation, we want to learn how *likely* new data is. Is a 2 m tall 16 year old more or less likely than a 1.5 m tall 80 year old?

The number that the model produces should be a *probability* if the feature spaces is discrete. That means that the sum of all answers over the whole feature space should be 1.

If the feature space is continuous (numeric features), the answer should be a probability *density* (and all answers should *integrate* to 1).



Density estimation is the task of modelling the *probability distribution* behind your data.

Here is an example: the final grades from 2017. If you know a bit of statistics, you can probably see sort of a normal distribution in this. Once you've fitted a normal distribution, you can give a density estimate for any grade.

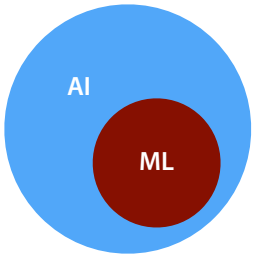
Except that there are three peaks. these could be explained by noise, but we could also fit a mixture of three normal distributions to this data, to explain the peaks. This is a much more difficult task, to which we will return in a few weeks.

With complex data, it's often easier to *sample* from a probability distribution that it is to get a probability (density) estimate. Building a model from which you can sample new examples is called **generative modelling**.

These people *don't exist*. These pictures were *sampled* from a model trained on a large dataset of images of faces. Note that this is not a 3d model, or a generator that started with a basic face and filled in the details: all the model saw was a large collection of pictures. This is a typical example of the power of **deep learning**, which we will discuss in the third week.

image source: https://arxiv.org/abs/1812.04948

Machine Learning vs. AI



AI, but not ML: ~~playing chess~~, automated reasoning, planning.

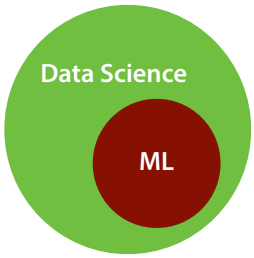
76

I've told you what machine learning is. Now let's look at what it isn't. How it relates to other fields and what the differences are.

First up, **Artificial Intelligence**. Machine Learning is a subfield of AI. If we want to make a general AI that can do everything we can, it needs to be capable of learning. But there are many other problems and fields in AI that have nothing to do with learning.

Other subfields, like natural language processing, are greatly helped by ML techniques, but can also be tackled without.

Machine Learning vs. Data Science



Data Science, but not ML: Gathering data, harmonising data, interpreting data.

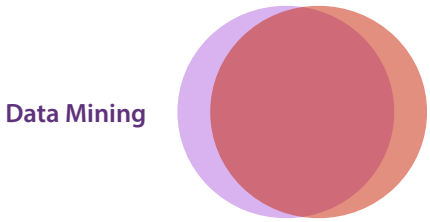
next week

77

Similarly all ML is Data Science, but not all Data Science is ML. Often, ML is used as part of a larger data science pipeline.

Thursday next week, we will look at some of the data science aspects required to perform Machine Learning experiments.

Machine Learning vs. Data Mining



More DM than ML: Finding common clickstreams in web logs. Finding fraud in transaction networks.
More ML than DM: Spam classification, predicting stock prices, learning to control a robot.

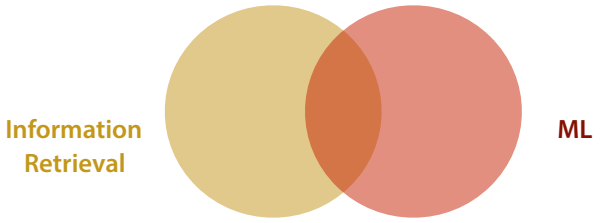
78

Here is a more subtle distinction: data **mining** and machine learning. Opinions differ, but I would characterise them as follows. Both analyze data, but they do so with different aims: machine learning aims to produce a *model* of the data, data mining aims to produce intelligence.

Another distinction is that machine learning focuses on prediction: trying to predict a target value for new data, whereas data mining tries to navigate and simplify the data so that it becomes useful for users.

If you have a dataset, but you expect never ever to see any new data from the same source, you can perform data mining on it, but performing machine learning on it is not much use (although your data mining will probably use machine learning *techniques*).

Machine Learning vs. Information Retrieval

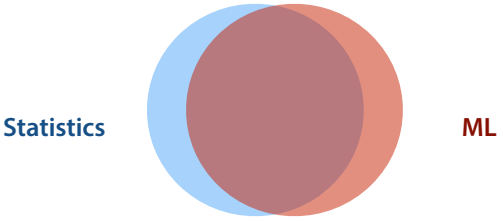


79

Information retrieval (building search engines) may look at first like a field that is completely distinct from ML. But on closer inspection, it turns out that you can model IR as a kind of classification task: your instances are documents, and your aim is to classify them into relevant or irrelevant (for a particular query).

This may seem a bit extreme, since the class imbalance is so high, but this has actually helped us in ML to think more clearly about problems with high class-imbalance (where ranking is a more appropriate way to think about the task than classification).

Machine Learning vs. Statistics



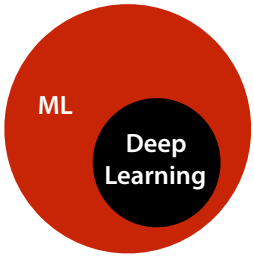
Stats but not ML: Analyzing research results. Experiment design. Courtroom evidence.
More ML than Stats: Spam classification, movie recommendation,.

80

Here is another subtle distinction. Statistics and ML both focus on analysing data, and modelling it in some way. In fact, many of the models we use in ML were invented by statisticians before the name Machine Learning existed. The distinction isn't always clear, but the most important difference is that Statistics aim to get at the *truth*, whereas machine learning tries to come up with something that *works*, regardless of whether it's true.

Consider spam classification. We usually model emails as a bag of independently drawn words. This has nothing to do with the way emails are actually written. Still, it works well enough to let people control their in-box. Contrast this with proving in a courtroom that a particular piece of DNA evidence really puts a suspect at the scene of the crime. Here, we're interested in more than just getting a useful model that captures some of the data, we want the truth.

Machine Learning vs. Deep Learning



starting in week 4

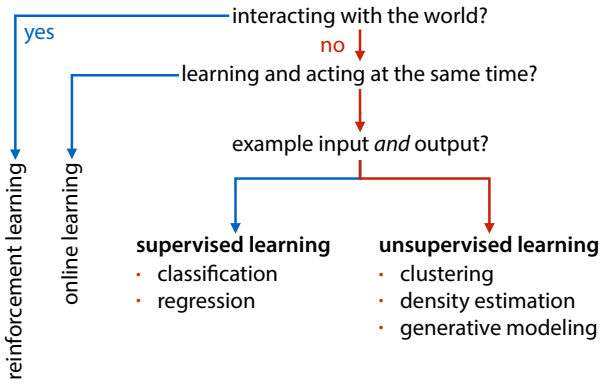
81

Finally, you may have heard a lot in the news in recent years about deep learning and wondered whether it is the same as machine learning or something different, or what. Here I can be clear. Deep Learning is a subfield of Machine Learning. All Deep Learning is Machine Learning but not all Machine Learning is Deep Learning.

We will discuss Deep Learning, and what makes it so special at various points in the course.

---

summary



82

---

offline machine learning: the basic recipe

Abstract (part of) your problem to a **standard task.**
Classification, Regression, Clustering, Density estimation, Generative Modeling, Online learning, Reinforcement Learning, Structured Output Learning

Choose your **instances** and their **features**.
For supervised learning, choose a target.

Choose your **model class**.
Linear models, Decision Trees, kNN,

Search for a good model.
Usually, a model comes with its own search method. Sometimes multiple options are available.

83

To summarize: this is the basic recipe for doing machine learning. It doesn't always fit the problem, and we'll look at those cases too. Sometimes your problem doesn't fit very well (reinforcement learning, recommendation), sometime deviating slightly can help performance (sequence learning) and sometimes, taking a radically different approach can turn everything on its head (deep learning).

Nevertheless, if you want to keep things simple, and use existing solutions, **the basic recipe** is a good starting point,

summary

abstracting your problem: instances, features, target.

supervised learning: classification, regression.

- linear models, tree models, NN models

unsupervised learning: clustering, density estimation, generative modeling.

loss function: maps a choice of model to a *loss* for the current data (the lower the better).

overfitting. Test and training data.

84

week 1
Introduction
Classification, Regresssion, Clustering
Linear Models 1
Hyperplanes, Random Search, Gradient descent

week 2
Method 1
Comparing methods
Method 2
preprocessing
missing values, outliers, dim. reduction

week 3
Probability 1
Gaussians, (Naive) Bayes
Entropy, Logistic regression
Linear Models 2
Neural nets, SVMs

week 4
Deep Learning 1
SGD
Backpropagation
CNNs
Probability 2
Expectation Maximization
pick a topic!

week 5
Deep Learning 2
Generative models:
GANs, VAEs
Tree Models & ensembles
Decision trees
Boosting, Bagging

week 6
Sequences
Markov models, Word2Vec, RNNs
Matrix models
Recommender systems
PCA revisited

week 7
Reinforcement Learning
Q learning, Policy gradients, **AlphaStar**
Review
Open problems
Responsible ML

week 8
exam
Monday 23 March

project deadline
Friday 27 March

85

what to do now

- Register for a workgroup (optional)
- Get a group together, or register for a random project group.
- First worksheet
- First homework

86

HAPPY LEARNING!

<- go easy on me!

mlcourse@peterbloem.nl

91

If you have any questions, please ask them on the Canvas discussion board if at all possible. If it's a personal question, or if you need to ask me personally for some other reason, please don't hesitate to email.